```
package KnowledgeAgents;

import db.*;
import gui.*;
import readers.*;
import se.*;
import text.*;
import utils.*;
import ha.*;

import java.awt.*;
import java.util.*;
import java.io.*;

/**
 * The class which implements a Knowledge Agent
 */
public class Agent implements Runnable
{
  interface Action
  {
    void perform(Agent agent);
  }
  static class ActionExit implements Action
  {
    public void perform (Agent agent)
    {
      agent.exit();
    }
  }
  static class ActionRefineQuery implements Action
  {
    String query;
    int    queryXFactor;
    TextField tfQuery;

    ActionRefineQuery (String query,
                       int    queryExpandFactor,
                       TextField tfQuery)
    {
      this.query           = new String(query);
      this.queryXFactor    = queryExpandFactor;
      this.tfQuery         = tfQuery;
    }

    public void perform (Agent agent)
    {
      agent.refineQuery(query, queryXFactor,tfQuery);
    }
  }
  static class ActionTextQuery implements Action
  {
    String query;
    int    queryXFactor;
    int    rootSetSize;
    int    expandFactor;
    int    numResults;
    float  maxAnchorWeight;
    String outputFileName;
    SearchEngine[]  engines;
    boolean  update;
```

```
    ActionTextQuery (String query,
                     int    queryExpandFactor,
                     int    rootSetSize,
                     int    expandFactor,
                     int    numResults,
                     float  maxAnchorWeight,
                     String outputFileName,
                     SearchEngine[] engines,
                     boolean  update)
    {
      this.query           = new String(query);
      this.queryXFactor    = queryExpandFactor;
      this.rootSetSize     = rootSetSize;
      this.expandFactor    = expandFactor;
      this.numResults      = numResults;
      this.maxAnchorWeight = maxAnchorWeight;
      this.outputFileName  = new String(outputFileName);
      this.engines         = engines;
      this.update         = update;
    }

    public void perform (Agent agent)
    {
      agent.textQuery (query,
                       queryXFactor,
                       rootSetSize,
                       expandFactor,
                       numResults,
                       maxAnchorWeight,
                       outputFileName,
                       engines,
                       update);
    }
  }
  static class ActionLinkQuery implements Action
  {
    String[] rootNames;
    int      expandFactor;
    int      numResults;
    float    maxAnchorWeight;
    String   outputFileName;
    SearchEngine[]  engines;
    boolean  update;

    ActionLinkQuery (String query,
                     int    expandFactor,
                     int    numResults,
                     float  maxAnchorWeight,
                     String outputFileName,
                     SearchEngine[] engines,
                     boolean update)
    {
      StringTokenizer stk = new StringTokenizer(query);
      int             ntk = stk.countTokens();
      this.rootNames      = new String[ntk];
      try
      {
      while (ntk > 0)
        rootNames [--ntk] = stk.nextToken();
      }
```

```
        catch (NoSuchElementException e){/* can't happen */}

125     this.expandFactor    = expandFactor;
        this.numResults      = numResults;
        this.maxAnchorWeight = maxAnchorWeight;
        this.outputFileName  = new String(outputFileName);
        this.engines         = engines;
130     this.update          = update;
      }

      public void perform (Agent agent)
      {
135     agent.linkQuery (rootNames,
                         expandFactor,
                         numResults,
                         maxAnchorWeight,
                         outputFileName,
140                      engines,
                         update);
      }
    }
    static class ActionAddSites implements Action
145 {
      String[] addNames;

      ActionAddSites (String names)
      {
150     StringTokenizer stk = new StringTokenizer(names);
        int             ntk = stk.countTokens();
        this.addNames       = new String[ntk];
        try
        {
155     while (ntk > 0)
          addNames [--ntk] = stk.nextToken();
        }
        catch (NoSuchElementException e){/* can't happen */}
      }
160
      public void perform (Agent agent)
      {
        agent.repository.forceAddSites(addNames);
      }
165 }
    // data members for this class
    private AgentManager boss;
    private ReadManager  reader;
    private SEManager    seManager;
170 private AgentGUI     gui;
    private boolean      cont;
    private Action       action;
    private ServiceQueue servQ;
    private Trace        trace;
175 private IndexUtil    indexUtil;
    private Repository   repository;

    // how many read requests are allowed to wait in the Service Queue
    private static final int REQ_FACTOR = 2;
180
    // The minimal portion of the text score in the final ranking of sites
    private static final double MIN_TEXT_PORTION = 0.1;
```

```
    /**
185  * Dummy constructor - will be changed entirely.
     * @param The <code>AgentManager</code> who owns this <code>Agent</code>.
     * @param The <code>ReadManager</code> which serves this <code>Agent</code>.
     * @param Input stream for reading the state of an existing Agent.
     */
190 public
    Agent (AgentManager    manager,
         ReadManager     reader,
         IndexUtil       indexUtil,
         FileInputStream fis       )
195 {
      // initialize private members according to input args
      this.boss      = manager;
      this.reader    = reader;
      this.indexUtil = indexUtil;
200   this.cont      = true;

      // allocate private objects
      seManager   = new SEManager();
      trace       = new Trace();
205   servQ       = new ServiceQueue(reader,trace);
      gui         = null;
      // split according to the newOrExist argument.
      if ( null != fis )
      {
210     try
        {
        ObjectInputStream ois = new ObjectInputStream(fis);
        restoreState(ois);
        }
215     catch (Exception e)
        {
        System.err.println("While Reading Agent's state:\n"+e);
        }
      }
220   else
        repository = new Repository(this,trace);
    }
    /**
     * starts the user interface and waits for commands.
225  */
    public void
    run ()
    {
      // check that the repository has finished initializing
230   if ( repository.getName() == null )
      {
        try
        {
        synchronized(this) {
235       wait();
        }
        }
        catch (InterruptedException e) {}
      }
240
      // init the GUI, tell the ServiceQueue about the GUI as well
      gui = new AgentGUI(this, trace);
      gui.pack();
      gui.setSize(700,600);
```

```
245       gui.setTitle(repository.getName() + " Knowledge Agent");
          gui.show();
          servQ.setGUI(gui);
          if ( repository.initNames != null )
            updateRepSites(repository.initNames, true);
250
          while (cont)
          {
            gui.setPhase("Ready");
            try // wait for action requests
255         {
            synchronized(this) {
              wait();
            }
            }
260         catch (InterruptedException e) {}

            action.perform(this);
          }

265     }

        /**
         * Sets the action which the agent will execute next.
         * @param The action to be taken.
270      */
        synchronized public void
        setAction (Action action)
        {
          synchronized(this) {
275         this.action = action;
            notify();
          }
        }

280     /**
         * Causes the agent to free resources and stop executing.
         */
        public void
        exit()
285     {
          gui.dispose();
          cont = false;
        }
        /**
290      * Reads the collection which pertains to a given text query.
         *
         * @param The query.
         * @param Number of <b>LA<b>s to add to each query term.
         * @param Number of results to fetch from each search engine for the query.
295      * @param Number of incoming and outgoing links to/fro each root site
         *        to collect.
         * @param Number of best ranking sites to display.
         * @param The maximal anchor text weight allowed.
         * @param File where the results will be written.
300      * @param An array with the search engines to be used for this query.
         * @param TRUE if the user wants to update the repository
         *
         * The method uses the search engines to get a root set of sites which
         * match the queries. That set is expanded to the WWW-neighborhood of
305      * distance 1, and those sites are ranked.
```

```
 */
public void
textQuery (String query,
           int    queryXFactor,
           int    rootSetSize,
           int    expandFactor,
           int    numResults,
           float  maxAnchorWeight,
           String outputFileName,
           SearchEngine[]  engines,
           boolean update)
{
  SiteDB siteDB  = new SiteDB(trace,gui);// the Site Database for this query
  Vector root    = new Vector();         // the root sites for this query
  Vector nonRoot = new Vector();         // the non-root sites for this query

  gui.setNSites(0);
  gui.setPhase("Reading root sites");

  MiniProfile miniProf = new MiniProfile();
  String      xq       = repository.expandQuery(query,
                                                queryXFactor,
                                                miniProf,
                                                indexUtil,
                                                trace     );
  String[] queries = {xq};
  Filter[] filters = {new Filter(siteDB,SiteEntry.GROUP_ROOT)};
  seManager.query (engines,
                   queries,
                   rootSetSize,
                   servQ,
                   filters,
                   false,
                   root,
                   trace             );

  // incorporate repository sites
  repository.incorporate(siteDB,nonRoot);

  // collect backward set of root
  int resPerEngine = expandFactor/engines.length;
  if ( resPerEngine*engines.length < expandFactor )
    resPerEngine++;

  if ( resPerEngine > 0 )
  {
    gui.setPhase("Collecting backward links");
    getBackSet (siteDB,
                engines,
                root,
                resPerEngine,
                nonRoot,
                SiteEntry.GROUP_BROOT);
  }

  // read root sites, collect forward set of root, score lexical.
  gui.setPhase("Collecting forward links");
  getForwSet(siteDB,
             root,
             expandFactor,
             nonRoot,
```

```
                SiteEntry.GROUP_FROOT,
                miniProf,
                true                    );
370
        // resolve previously unresolved forward root links
        gui.setPhase("Resolving unresolved links.");
        resolveUnresolved(siteDB,
                        root    );
375
        // read all non-root sites, resolve links, score lexical.
        gui.setPhase("Reading non-root sites.");
        getForwSet(siteDB,
                nonRoot,
380             0,
                null,
                SiteEntry.GROUP_DUMMY,
                miniProf,
                true       );
385     /*
         * now we have a graph, and a lexical score for each site.
         * get link-based scores.
         */
        rankSites(query,
390             rootSetSize,
                expandFactor,
                numResults,
                maxAnchorWeight,
                outputFileName,
395             xq,
                miniProf,
                siteDB,
                update);
    }
400 /**
     * Reads the collection which pertains to a given link query.
     *
     * @param An array containing the names of URLs given by the user.
     * @param Number of incoming and outgoing links to/fro each root site
405  *        to collect.
     * @param Number of best ranking sites to display.
     * @param The maximal anchor text weight allowed.
     * @param File where the results will be written.
     * @param An array with the search engines to be used for this query.
410  * @param TRUE if the user wants to update the repository
     *
     * The method expands the root set of sites which the user has supplies.
     * The set of expanded sites is then ranked.
     */
415 public void
    linkQuery (String[] rootNames,
            int      expandFactor,
            int      numResults,
            float    maxAnchorWeight,
420         String   outputFileName,
            SearchEngine[]  engines,
            boolean  update)
    {
      SiteDB siteDB  = new SiteDB(trace,gui); // the Site Database for this query
425   Vector root    = new Vector();
      Vector bRoot   = new Vector();
      Vector fRoot   = new Vector();
```

```
        Vector others  = new Vector();
        MiniProfile miniProf = new MiniProfile();
430
        gui.setNSites(0);
        gui.setPhase("Initializing the database");
        initRootByNames(rootNames,
                        root,
435                     siteDB    );

        // collect backward set of root
        int resPerEngine = expandFactor/engines.length;
        if ( resPerEngine*engines.length < expandFactor )
440       resPerEngine++;

        if (resPerEngine > 0)
        {
          gui.setPhase("Collecting backward links of Root Set");
445       getBackSet (siteDB,
                  engines,
                  root,
                  resPerEngine,
                  bRoot,
450               SiteEntry.GROUP_BROOT );
        }

        // read root sites, collect forward set of root, score lexical.
        gui.setPhase("Collecting forward links of Root Set");
455     getForwSet(siteDB,
                 root,
                 expandFactor,
                 fRoot,
                 SiteEntry.GROUP_FROOT,
460              miniProf,
                 false                );

        // now assign weights to the terms in the MiniProfile according to the
        // accumulated profile.
465     repository.assignWeights(miniProf);
        // incorporate repository sites
        repository.incorporate(siteDB,others);

        // collect backward set of fRoot
470     if ( resPerEngine > 0 )
        {
          gui.setPhase("Collecting backward links of Forward Root Set");
          getBackSet (siteDB,
                  engines,
475               fRoot,
                  resPerEngine,
                  others,
                  SiteEntry.GROUP_BF );
        }
480
        // read bRoot sites, collect their forward set.
        gui.setPhase("Collecting forward links of Back-of-Root Set");
        getForwSet(siteDB,
                 bRoot,
485              expandFactor,
                 others,
                 SiteEntry.GROUP_FB,
                 miniProf,
```

```
                true              );
490
        // resolve previously unresolved forward root links
        gui.setPhase("Resolving unresolved links.");
        resolveUnresolved(siteDB, root);
        resolveUnresolved(siteDB, bRoot);
495
        // read all other sites, resolve links, score lexical.
        gui.setPhase("Reading forward-root sites.");
        getForwSet(siteDB,
                fRoot,
500             0,
                null,
                SiteEntry.GROUP_DUMMY,
                miniProf,
                true                );
505     gui.setPhase("Reading other previously unread sites.");
        getForwSet(siteDB,
                others,
                0,
                null,
510             SiteEntry.GROUP_DUMMY,
                miniProf,
                true                );

        // fix the lexical score of the root sites.
515     fixRootTextScores (siteDB, root);
        /*
         * now we have a graph, and a lexical score for each site.
         * get link-based scores.
         */
520     // first, build a String representation of this Link Query.
        StringBuffer queryString = new StringBuffer("(Link Query)<BR>");
        for ( int i=0; i<rootNames.length; i++ )
        {
          queryString.append(rootNames[i]);
525       queryString.append("<BR>");
        }

        rankSites(queryString.toString(),
                rootNames.length,
530             expandFactor,
                numResults,
                maxAnchorWeight,
                outputFileName,
                null,
535             miniProf,
                siteDB,
                update);
    }
    /**
540  * Calculates the link rankings and combined rankings of the sites.
     *
     * @param The query that was run.
     * @param Size of the root set.
     * @param Link Expansion factor.
545  * @param Number of top ranking sites requested.
     * @param The maximal anchor text weight allowed.
     * @param Output file.
     * @param The expanded query.
     * @param The <code>MiniProfile</code> by which to weigh links.
```

```
     * @param The Site Database collected in the backend stages.
     * @param TRUE if user wants to update the repository
     */
    private void
    rankSites (String       query,
               int          rootSetSize,
               int          expandFactor,
               int          numResults,
               float        maxAnchorWeight,
               String       outputFileName,
               String       xq,
               MiniProfile  miniProf,
               SiteDB       siteDB,
               boolean      update)
    {
      // get an array of sites (enumerate the sites), sort resolved links
      int nSites = siteDB.nSites();
      gui.setPhase("Sorting resolved links.");
      SiteEntry[] sites = getSiteArray(nSites, siteDB.sites());
      sortScoreResolved(sites, miniProf, maxAnchorWeight);

      // build H&A matrices, score, sort and extract best sites!!!
      gui.setPhase("Building matrices and ranking");
      double[] authorityWeights, authoritySTWeights;
      double[] hubWeights,       hubSTWeights;
      double[] textWeights;

      if (nSites < numResults)
        numResults = nSites;

      /*
       * out degree rankings, debug
      double[] degWeights = new double[sites.length];
      double ssq=0;
      for ( int ttt = 0; ttt < sites.length; ttt++ )
      {
        degWeights[ttt] = sites[ttt].getOutDegree();
        ssq += degWeights[ttt]*degWeights[ttt];
      }
      ssq = Math.sqrt(ssq);
      for ( int ttt = 0; ttt < sites.length; ttt++ )
        degWeights[ttt] /= ssq;
      int[] topDeg = Heap.getBest(degWeights, numResults);
      *
      */
      textWeights = normalizeTextWeights(sites);
      int[] topTextInd = Heap.getBest(textWeights,numResults);

      {
        MRMat authorityMat = new MRMat(sites,true,trace);
        authorityWeights   = authorityMat.power();
      }
      int[] topAuthInd     = Heap.getBest(authorityWeights,numResults);

      {
        MRMat hubMat = new MRMat(sites,false,trace);
        hubWeights   = hubMat.power();
      }
      int[] topHubInd= Heap.getBest(hubWeights,numResults);

      {
```

```
        STMat authorityMat = new STMat(sites,true,trace);
        authoritySTWeights = authorityMat.power();
      }
      int[] topSTAuthInd   = Heap.getBest(authoritySTWeights,numResults);
615   {
        STMat hubMat = new STMat(sites,false,trace);
        hubSTWeights = hubMat.power();
      }
      int[] topSTHubInd= Heap.getBest(hubSTWeights,numResults);
620
      double[] combWeights = genCombWeights(authorityWeights,
                                  hubWeights,
                                  authoritySTWeights,
                                  hubSTWeights,
625                               textWeights,
                                  rootSetSize,
                                  (xq == null)       );
      int[] topComb = Heap.getBest(combWeights,numResults);

630   OutStruct[] outStructs = {
        trace.isLit(Trace.PRINT_SITES)    ?
        new OutStruct("EntireDB",
                  "All Sites",
                  combWeights,
635               (int[]) null       ) : null,

        new OutStruct("Overall",
                  "Overall Rankings",
                  combWeights,
640               topComb),

        new OutStruct("MRauths",
                  "Mutual Reinforcement top Authorities",
                  authorityWeights,
645               topAuthInd),

        new OutStruct("MRhubs",
                  "Mutual Reinforcement top Hubs",
                  hubWeights,
650               topHubInd  ),

         new OutStruct("STauths",
                  "Stochastic top Authorities",
                  authoritySTWeights,
655               topSTAuthInd),

        new OutStruct("SThubs",
                  "Stochastic top Hubs",
                  hubSTWeights,
660               topSTHubInd  ),
        /*
        new OutStruct("OutDeg",
                  "OutDegree rankings",
                  degWeights,
665               topDeg  ),
        */
        new OutStruct("Text",
                  "Text-Rich Sites",
                  textWeights,
670               topTextInd  )
      };
```

```
        // write output!!!
        gui.setPhase("Writing output.");
675     writeOutput (query,
                     xq,
                     rootSetSize,
                     expandFactor,
                     numResults,
680                  outputFileName,
                     outStructs,
                     sites           );

        // now update the repository
685     if (update)
           repository.transfuse(sites,combWeights);
      }

      /**
690    * Finds the URLs which point into sites from a root set of URLs.
       *
       * @param The <code>SiteDB</code> object for this mission.
       * @param A stack with the search engines to be used for this mission.
       * @param A Vector of root-site names.
695    * @param Maximal number of incoming links from each root site to collect.
       * @param The <code>Vector</code> where the results will be stored.
       * @param The group of sites which will be added through this operation.
       */
      private void
700   getBackSet (SiteDB siteDB,
                  SearchEngine[] engines,
                  Vector root,
                  int    expandFactor,
                  Vector backSet,
705               int    group       )
      {
        String[]     siteNames = new String[root.size()];
        PairFilter[] filters   = new PairFilter[root.size()];

710     for (int i=0; i < root.size(); i++)
        {
          try {
          siteNames[i] = (String) root.elementAt(i);
          filters[i]   = new PairFilter(siteDB, siteNames[i], group);
715       }
          catch (NullContextException e) { /* shouldn't happen */ }
        }

        seManager.query( engines,
720                      siteNames,
                         expandFactor,
                         servQ,
                         filters,
                         true,
725                      backSet,
                         trace          );
        return;
      }
      /**
730    * Reads and parses a set of URLs, scores them,
       *       and collects their outgoing links.
       *
```

```
     * @param The <code>SiteDB</code> object for this mission.
     * @param A stack with the search engines to be used for this mission.
     * @param A Vector of root-site names.
     * @param Maximal number of outgoing links from each root site to collect.
     * @param The <code>Vector</code> where the outgoing links will be stored.
     * @param The group of sites which will be added through this operation.
     * @param The <code>MiniProfile</code> object to update/score with.
     * @param Score read sites if <code>true</code>, update
     *        <code>MiniProfile</code> if <code>false</code>.
     */
    private void
    getForwSet (SiteDB      siteDB,
                Vector      root,
                int         expandFactor,
                Vector      forwSet,
                int         group,
                MiniProfile miniProf,
                boolean     doScore  )
    {
      Enumeration rootSites = root.elements();
      int         pendingReads = root.size();
      ReadAns     ans;
      int         numReaders   = reader.getNumReaders();
      int         pendingReqs  = 0;

      trace.write("FRS *** Getting Forward Set ***", Trace.FORWARD_SET);
      trace.write("FRD Free Readers: "+reader.getNumFreeReaders(),
              Trace.FREE_READERS                                    );

      // enter the first batch of reading requests
      pendingReqs = insertReqs(rootSites,pendingReqs,numReaders);

      // now wait to collect all of the results
      while (pendingReads > 0)
      {
        gui.setLeft(pendingReads);
        synchronized(servQ)
        {
        if ( servQ.noMoreAnswers() )
          {
          try {
            servQ.wait();
          }
          catch (InterruptedException e) {}
        }
        }
        ans = servQ.getFirstAnswer();
        try
        {
        String readUrlName = ans.req.urlString;
          trace.write("FRS\tworking on "+ readUrlName, Trace.FORWARD_SET);
        pendingReads--;
        if ( --pendingReqs == numReaders)
          pendingReqs = insertReqs(rootSites,pendingReqs,numReaders);

        // resolve the answer into a SiteEntry, get a filter object for it.
        SiteEntry  curSite = siteDB.getEntry(readUrlName);

        // check that the return code is ok.
        if ( ans.rc != WebReader.WEBREAD_BAD  &&
             ans.rc != WebReader.WEBREAD_EMPTY   )
```

```
        {
795       curSite.setReadStat(ans.rc == WebReader.WEBREAD_OK ?
                              SiteEntry.READSTAT_YES :
                              SiteEntry.READSTAT_PARTIAL      );

          // get a filter for the current URL, and parse its contents.
800       PairFilter filter  = new PairFilter(siteDB, readUrlName, group);
          HTMLParse parsed = new HTMLParse(ans.contents, trace);
          /*
           * save the title and treat the links:
           * add upto ExpandFactor new sites, resolve links, keep unresolved.
805        */
          curSite.setTitle(parsed.title);
          Enumeration eLinks   = parsed.links.elements();
          int         newSites = 0;
          while (eLinks.hasMoreElements())
810       {
            SiteLinks.UnresolvedLink link =
              (SiteLinks.UnresolvedLink)eLinks.nextElement();
              trace.write("FRS\t\t"+link.destName + ' ' + link.anchorText,
                   trace.FORWARD_SET_LINK_DETAILS);
815         //link.destName = filter.URLInContext(link.destName);
            //if (link.destName != null)
            //{
              // are we allowed to add new sites?
              if ( newSites < expandFactor )
820           {
              if ( filter.addSiteAndLink(link) == Filter.NEW )
              {
                // add the new site to the forward set, and count it
                forwSet.addElement(link.destName);
825             newSites++;
                trace.write("FRS\t\tAdding new site # " + newSites +
                          " : " + link.destName,
                          trace.FORWARD_SET_LINK_DETAILS        );
              }
830           }
              else
              filter.keepSiteAndLink(link);
            //}
          }
835       if ( doScore )
          {
            trace.write("SCR_HTML Scoring "+readUrlName,Trace.SCORE_SITES);
            curSite.textScore = indexUtil.scoreHTMLPage(parsed,
                                                miniProf,
840                                             trace);
            trace.write("SCR_HTML Score="+curSite.textScore,Trace.SCORE_SITES);
          }
          else
            indexUtil.updateProfile(parsed,miniProf,true);
845
        } // end of treating a well-read site
        else
          curSite.setReadStat(SiteEntry.READSTAT_PROBLEM);

850     }   // end of read-answer resolution
        catch (NullContextException e)
        {
        ans.rc = WebReader.WEBREAD_BAD;
        System.err.println(e);
```

```
855         }
            catch (NullPointerException e)
            {
            trace.write("FRS\t"+e,Trace.FORWARD_SET);
            System.err.println(e);
860         if (ans == null)
              trace.write("FRS\tans is bad!!!",Trace.FORWARD_SET);
            else
              ans.rc = WebReader.WEBREAD_BAD;
            }
865       } // end of loop over pending reads
          gui.setLeft(0);
          return;
        }
        /**
870      * Reads a set of URLs, and updates the repository with their contents.
         *
         * @param The array of URL names to read.
         * @param <code>true</code> to add URLs, <code>false</code> to remove them.
         */
875     void
        updateRepSites (String[] names,
                        boolean  add   )
        {
          ReadAns     ans;
880       int         pendingReqs;

          gui.setPhase( add ? "Adding Sites to Knowledge Base" :
                              "Removing Sites from Knowledge Base");
          trace.write("REP *** Reading sites for repository ***", Trace.REPOSITORY);
885       trace.write("FRD Free Readers: "+reader.getNumFreeReaders(),
                  Trace.FREE_READERS                                    );

          // enter the reading requests
          for ( pendingReqs = 0;
890           pendingReqs < names.length && names[pendingReqs] != null;
              pendingReqs++ )
          {
            RepReadReq req = new RepReadReq(names[pendingReqs], servQ, add);
            servQ.addRequest (req);
895       }

          // now wait to collect all of the results
          while (pendingReqs > 0)
          {
900         gui.setLeft(pendingReqs);
            /*
            if ( servQ.noMoreAnswers() )
            {
            try {
905           synchronized(servQ){
                servQ.wait();
              }
            }
            catch (InterruptedException e) {}
910         }
            */
            synchronized(servQ)
            {
            if ( servQ.noMoreAnswers() )
915           {
```

```
            try {
              servQ.wait();
            }
            catch (InterruptedException e) {}
        }
        }
        ans = servQ.getFirstAnswer();
        pendingReqs--;
        if ( ans != null )
        {
        String readUrlName = ans.req.urlString;
          trace.write("REP\tworking on "+ readUrlName, Trace.REPOSITORY);

        // check that the return code is ok.
        if ( ans.rc != WebReader.WEBREAD_BAD  &&
             ans.rc != WebReader.WEBREAD_EMPTY   )
        {
          HTMLParse parsed = new HTMLParse(ans.contents, trace);
          repository.updateSite(parsed, ((RepReadReq)ans.req).add, indexUtil);
        } // end of treating a well-read site
        }   // end of read-answer resolution
      } // end of loop over pending reads
      gui.setLeft(0);
      return;
    }
    /**
     * Resolves some of the previously unresolved outgoing links of a set of
     * sites.
     * @param The group of sites.
     */
    private void
    resolveUnresolved (SiteDB siteDB,
                   Vector siteVec)
    {
      Enumeration sites = siteVec.elements();

      while (sites.hasMoreElements())
      {
        SiteEntry site = (SiteEntry)
                       siteDB.getEntry((String)sites.nextElement());
        site.getLinks().resolveUnresolved(siteDB, site.getHostEntry());
      }
    }
    /**
     * Writes the output of a query, and displays it in a browser.
     *
     * @param The query that was run.
     * @param The expanded query.
     * @param Size of the root set.
     * @param Link Expansion factor.
     * @param Number of top ranking sites requested.
     * @param Output file.
     * @param The array containing the groups or rankings to print.
     * @param The results.
     */
    private void
    writeOutput (String       query,
                 String       xq,
                 int          rootSetSize,
                 int          linkExpansionFactor,
                 int          numResults,
```

```
                String       outputFileName,
                OutStruct[]  outStructs,
                SiteEntry[]  sites              )
980     {
          FileOutputStream out = null;

          if (trace.isLit(Trace.DUMP_SITEDB))
            dumpGraph(sites,query);
985
          try
          {
            // open a stream to the input file.
            out = new FileOutputStream(outputFileName);
990
            // prepare the title of the HTML page.
            outWrite(out,"<HTML><HEAD>\n<TITLE>"+
                    repository.getName() + " Knowledge Agent: Results" +
                  "</TITLE></HEAD>\n");
995         outWrite(out,"<BODY>\n");
            // prepare anchor links
            outWrite(out,"<P><FONT SIZE += 1>\n");
            outWrite(out,"<A HREF=\"#echo\">Argument Echo</A><BR>");
            // prepare an anchor link for each output structure
1000        for ( int i = 0; i < outStructs.length; i++ )
            {
            if ( null != outStructs[i])
              outWrite(out,   "<A HREF=\"#"       +
                     outStructs[i].anchorName + "\">" +
1005                 outStructs[i].anchorText + "</A><BR>\n" );
            }
            outWrite(out,"</FONT></P>\n");

            // Echo the invocation arguments
1010        outWrite(out,"<H2><A NAME=\"echo\">Invocation Arguments</A></H2>\n");
            outWrite(out,"<UL>");
            outWrite(out,"\n<LI><I>Query: </I>"+query);
            if ( xq != null )
            outWrite(out,"\n<LI><I>Expanded Query: </I>"+xq);
1015        outWrite(out,"\n<LI><I>Root Set Size: </I>"+rootSetSize);
            outWrite(out,"\n<LI><I>Link Expansion Factor: </I>"+linkExpansionFactor);
            outWrite(out,"\n<LI><I>Number of Sites in Collection: </I>"+
                   sites.length);
            outWrite(out,"\n</UL>\n");
1020
            // Write the results of each output structure
            for ( int j = 0; j < outStructs.length; j++ )
            {
            if ( null != outStructs[j] )
1025        {
              outWrite(out,"<H2><A NAME=\""     +
                     outStructs[j].anchorName + "\">" +
                     outStructs[j].anchorText + "</A></H2>");
              outWrite(out,"\n<OL>\n");
1030          for ( int i = 0; i < outStructs[j].indices.length; i++ )
              {
                int    iSite    = outStructs[j].indices[i];
                outWrite(out,"\n<LI>\n\t<A HREF=\""
                       +sites[iSite].getNormName()+"\">"
1035                   +sites[iSite].toString()+"</A>");
                outWrite(out," "+sites[iSite].getAttributeDesc());
                outWrite(out,"<BR>\n\t<B>Title:</B> "
```

```
                    +sites[iSite].getTitle());
              outWrite(out,"<BR>\n\t<B>Weight:</B> "+
                    Double.toString(outStructs[j].weights[iSite]) );
            }
            outWrite(out,"\n</OL>\n");
          }
          }

          // end the HTML page
          outWrite(out,"</BODY></HTML>\n");
        }
        catch (IOException e)
        {
          System.err.println (e);
        }
        catch (SecurityException e)
        {
        }
        finally
        {
          try {
          if ( null != out )
            out.close();
          } catch(Exception e){}
        }
        // open a browser with this file.
        BrowserControl.displayURL("file://"+outputFileName);

        return;
      }
      /**
       * Refines a query and writes the output to the standard output.
       * @param The query to refine.
       * @param Number of LAs to add to each term.
       */
      public void
      refineQuery (String query,
                   int    queryXFactor,
                   TextField tfQuery)
      {
        MiniProfile miniProf = new MiniProfile();
        String      xq       = repository.expandQuery(query,
                                          queryXFactor,
                                          miniProf,
                                          indexUtil,
                                          trace     );
        // Aya 20.10.99 - write the refined query into the query text field
        tfQuery.setText(xq);
        System.out.println("Refined Query: "+xq);
      }
      /**
       * Writes a message in the file.
       * @param The <code>FileOutputStream</code> object.
       * @param The message to write to the file.
       */
      private final void
      outWrite (FileOutputStream out,
                String           s   )
      {
        try {
          out.write(s.getBytes());
```

```
        }
1100    catch (IOException e)
        {
          System.err.println(e);
        }

1105    return;
      }
      /**
       * Inserts read requests into the <code>ServiceQueue</code>.
       * @param An enumeration of the sites whose forward set is required.
1110   * @param The number of pending read requests in the Queue.
       * @param The number of readers available to the application.
       * @returns The number of pending requests after the insertion of new
       *          read requests.
       */
1115  private int
      insertReqs(Enumeration rootSites,
                 int         pendingReqs,
                 int         numReaders  )
      {
1120    // this is also a good spot to suggest garbage collection
        System.gc();
        while (rootSites.hasMoreElements() &&
             pendingReqs < REQ_FACTOR * numReaders)
        {
1125      ReadReq req = new ReadReq( (String)rootSites.nextElement(), servQ);
          servQ.addRequest (req);
          pendingReqs++;
        }
        return pendingReqs;
1130  }
      /**
       * Gets an array of sites from the <code>SiteDB</code> object.
       * @param The number of sites in the collection
       * @param An Enumeration of the sites, as returned by the
1135   *        <code>SiteDB</code> object.
       * @returns An array of <code>SiteEntry</code>s.
       */
      static SiteEntry[]
      getSiteArray (int         nSites,
1140              Enumeration siteCol)
      {
        SiteEntry[] sites   = new SiteEntry[nSites];
        while (siteCol.hasMoreElements())
        {
1145      SiteEntry site = (SiteEntry) siteCol.nextElement();
          sites[site.getSiteNumber()] = site;
        }
        return sites;
      }
1150  /**
       * Sorts the resolved links of all sites, scores them,
       * and builds the incoming links arrays.
       * @param An array of all sites in the collection.
       * @param The <code>MiniProfile</code> which holds weighted terms.
1155   * @param The maximal anchor text weight allowed.
       */
      private void
      sortScoreResolved (SiteEntry[] sites,
                         MiniProfile miniProf,
```

```
                    float       maxAnchorWeight)
    {
      for ( int i = 0; i < sites.length; i++ )
        sites[i].getLinks().sortScoreResolved(trace,
                                    indexUtil,
                                    miniProf,
                                    maxAnchorWeight);
    }
    /**
     * Initializes the SiteDB and the root Vector with an array of URL names.
     *
     * @param The array of URL names.
     * @param A <code>Vector</code> to hold the normalized URL names.
     * @param The <code>SiteDB</code> object.
     */
    private final void
    initRootByNames (String[] rootNames,
                 Vector   root,
                 SiteDB    siteDB    )
    {
      Filter filter = new Filter(siteDB, SiteEntry.GROUP_ROOT);

      for ( int i = 0; i < rootNames.length; i++ )
      {
        root.addElement(filter.normalize(rootNames[i]));
        filter.forceAddSite(rootNames[i]);
      }
      return;
    }
    /**
     * Fixes the text scores of root sites in the link-query path.
     * @param The <code>SiteDB</code> object.
     * @param A <code>Vector</code> with the names of the root sites.
     */
    private void
    fixRootTextScores (SiteDB siteDB,
                   Vector root   )
    {
      double      maxTextScore = 0;

      // first step - find max score
      Enumeration sites        = siteDB.sites();
      while (sites.hasMoreElements())
      {
        double ts = ((SiteEntry)sites.nextElement()).textScore;
        if ( maxTextScore < ts )
        maxTextScore = ts;
      }

      // second step - assign each root site the max score
      for ( int i=0; i < root.size(); i++ )
        siteDB.getEntry((String)root.elementAt(i)).textScore = maxTextScore;
    }
    /**
     * Builds a normailzed text weights array.
     * @param The array of sites.
     * @returns An array of normalized text weights.
     */
    private double[]
    normalizeTextWeights (SiteEntry[] sites)
    {
```

```
        double[] weights = new double[sites.length];
        double   sumSQ   = 0;
        int      i;

1225    for ( i=0; i < sites.length; i++ )
        {
          weights[i] = sites[i].textScore;
          sumSQ += sites[i].textScore * sites[i].textScore;
        }
1230    if ( sumSQ > 0 )
        {
          sumSQ = Math.sqrt(sumSQ);
          for ( i=0; i < sites.length; i++ )
          weights[i] /= sumSQ;
1235    }

        return weights;
      }
      private double[]
1240  genCombWeights (double[] authorityWeights,
                      double[] hubWeights,
                      double[] authoritySTWeights,
                      double[] hubSTWeights,
                      double[] textWeights,
1245                  double   rootSetSize,
                      boolean  isLinkQuery        )
      {
        int      nSites    = textWeights.length;
        double   invNSites = 1.0 / nSites;
1250    double[] comb      = new double[nSites];

        double[][] weights = {authorityWeights,
                              hubWeights,
                              authoritySTWeights,
1255                          hubSTWeights,
                              textWeights};
        double     denom   = 1.0;
        double     subtract= 0.0;

1260    // first, see how much weight goes to the text score
        int        linkExpand  = (int)(nSites / rootSetSize);
        double     textPortion = 1.0 - linkExpand* (isLinkQuery ? 0.025 : 0.05);
        if ( textPortion < MIN_TEXT_PORTION )
          textPortion = MIN_TEXT_PORTION;
1265
        // initialize the coefficients of the score components
        double[]   coeffs  = {(1.0-textPortion) * 0.375,
                              (1.0-textPortion) * 0.125,
                              (1.0-textPortion) * 0.375,
1270                          (1.0-textPortion) * 0.125,
                              textPortion                };

        // calculate averages, build array of coefficients, denom and subtract.
        int i;
1275    for ( i = 0; i < coeffs.length; i++ )
        {
          double avg = Repository.avg(weights[i]);
          double std = Math.sqrt(invNSites-avg*avg);

1280      subtract  += avg * coeffs[i] / std;
          denom     *= std;
```

```
          coeffs[i] /= std;
        }
        for ( i = 0; i < coeffs.length; i++ )
1285      coeffs[i] *= denom;

        // now calculate the overall scores of the sites.
        for ( i = 0; i < nSites; i++ )
        {
1290      comb[i] = coeffs[0] * weights[0][i];
          for ( int j = 1; j < coeffs.length; j++ )
          comb[i] += coeffs[j] * weights[j][i];

          comb[i] = comb[i] / denom - subtract;
1295    }
        return comb;
      }

      /** dumps the collection and its link-structure into an external format */
1300  private void
      dumpGraph (SiteEntry[] sites,
                 String      query)
      {
        int iSite;
1305    int nLinks = 0;
        gui.setPhase("Dumping Site Graph");

        try {
          FileWriter file = new FileWriter("./siteDB.dump");
1310      file.write ("\nQuery String: " + query);
          file.write ("\n\nNumber of Sites : " + sites.length+"\n");

          // Site #     : name (cat) \n
          for ( iSite = 0; iSite < sites.length; iSite++ )
1315      {
          file.write("\nSite # "+iSite+" : " + sites[iSite].toString());
          nLinks += sites[iSite].getLinks().getOutDegree();
          }

1320      file.write ("\n\n\nNumber of Links: " + nLinks + "\n");
          for ( iSite = 0; iSite < sites.length; iSite++ )
          {
          int[] destNums = sites[iSite].getLinks().getDestNums();
          file.write("\nOutgoing Links of Site # " +
1325                 iSite + "(" + destNums.length   + ")\n");
          for ( int e = 0; e < destNums.length; e++ )
            {
            file.write(String.valueOf(destNums[e]));
            file.write(' ');
1330      }
          }
          file.close();
        }
        catch (IOException e)
1335    {
          System.err.println(e);
        }
        return;
      }
1340  /** Writes the Agent's Repository to a stream (saving the Agent's state) */
      void
      saveState (ObjectOutputStream oos) throws IOException
```

```
        {
          gui.setPhase("Saving State...");
1345      oos.writeObject(repository);
          gui.setPhase("Ready");
        }
        /** Reads the Agent's Repository from a stream, restoring the Agent's state*/
        void
1350    restoreState (ObjectInputStream ois)
          throws IOException, ClassNotFoundException
        {
          if ( gui != null )
            gui.setPhase("Restoring State...");
1355      repository = (Repository)ois.readObject();
          repository.setTraceAndAgent(this,trace);
          if ( gui != null )
            gui.setPhase("Ready");
        }
1360  }
```